§4 Amortized Analysis

- Algorithmic Cost Analysis
  - Motivation Bit-Cost of Repeated Increment/Binary Add.
  - Amortized vs. Average vs. Worst-Case Analysis
- Fibonacci Heaps:
  - Relaxed Binomial Trees
  - ExtractMin and DecreaseKey
- Minimum Spanning Tree
  - Prim's Algorithm with Binomial vs. Fibonacci Heap
  - Kruskal's Algorithm: *Union-Find* data type
- Disjoint Set Data Structures
  - Fast and Slowly Growing Functions
  - Analysis of *Union-by-Weight*
  - Lazy Union-by-Rank with Path Compression

Bit-Cost of Repeated Increment

Cost := #bit flips when counting to n

```
000…000
000…001
000…010
000…011
000…100
000…101
   ……
111…111
```

One increment of value $<n$: $O(\log n)$

n increments: $\sum_{j<n} \log j = \Theta(n \cdot \log n)$ *over*estimate

Bit #0 flips n times,

bit #1 incurs $n/2$ flips, bit #2: $n/4$ flips

$\sum \leq 2n = \Theta(n)$: constant (2) "per" **inc**

Potential method of amortized analysis:

Let c_j := cost of j-th operation, $j=1..n$; Φ_j arbitrary

Φ_j := #**1**s in counter after j-th op. = before $(j+1)$-st op.

$$\sum_{1\leq j\leq n} c_j/n \;\leq\; \max_j (c_j+\Phi_j-\Phi_{j-1}) + (\Phi_0-\Phi_n)/n \;\leq 2$$

Repeated *Addition* of Binary Powers

Cost := #bit flips

Lemma: When adding a binary power 2^j to a natural number in binary, #bit flips + *difference* in #**1**s $\equiv 2$.

```
  010011111101011
+000000010000
```

Problem: Add $2^{j_1}+2^{j_2}+\ldots+2^{j_n}$.

Theorem: Repeatedly (n times) adding powers of 2 incurs a total of $\leq 2n$ bit flips: has *amortized* cost 2.

Potential method of amortized analysis:

Let c_j := cost of j-th operation, $j=1..n$; Φ_j arbitrary

Φ_j := #**1**s in counter after j-th op. = before $(j+1)$-st op.

$$\sum_{1\leq j\leq n} c_j/n \;\leq\; \max_j \left(c_j+\Phi_j-\Phi_{j-1}\right) + (\Phi_0-\Phi_n)/n \;\leq 2$$

Amortized/Average/Worst Case

Example: Amortized #bit flips, counting to n in binary

Def: Fix data structure&algorithms for abstract data type: new initialized, then any n calls to method(s)/operations;
$\Rightarrow$ *total* worst-case cost $C(n)$ $\Rightarrow$ **amortized cost** $=C(n)/n$.

inc **& dec** ?

- (Ordinary) worst-case: $\mathbf{max}_{|\underline{x}|\le n}\, c(\underline{x})$
- Average case: $\mathbf{sum}_{|\underline{x}|\le n}\, c(\underline{x}) / \#\{\, \underline{x}: |\underline{x}|\le n \,\}$

pessimist

- Expected case: randomized algorithms (later).

$c(\underline{x})$:= cost of *one* call on arguments/data contents $\underline{x}$

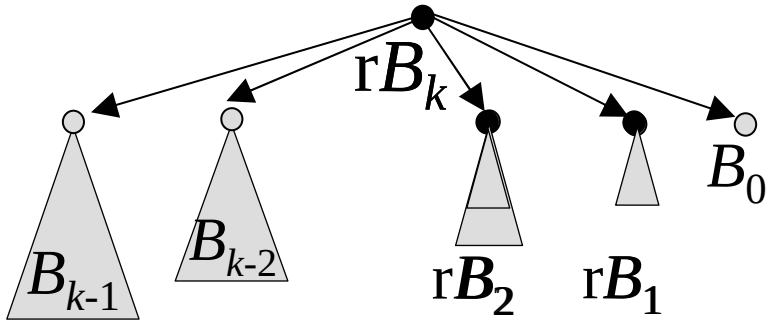
Potential method of amortized analysis:

Let c_j := cost of j-th operation, $j=1..n$; Φ_j arbitrary

Conceive Φ_j such that right hand side is „small“

$$\sum_{1\le j\le n} c_j/n \;\le\; \max_j (c_j+\Phi_j-\Phi_{j-1}) + (\Phi_0-\Phi_n)/n$$

Relaxed Binomial Trees



A *relaxed* binomial tree of order $k \geq 1$ is a root with k children: relaxed binomial trees of *distinct* orders.

Merge

Prune

from *un*marked parent: add mark to parent

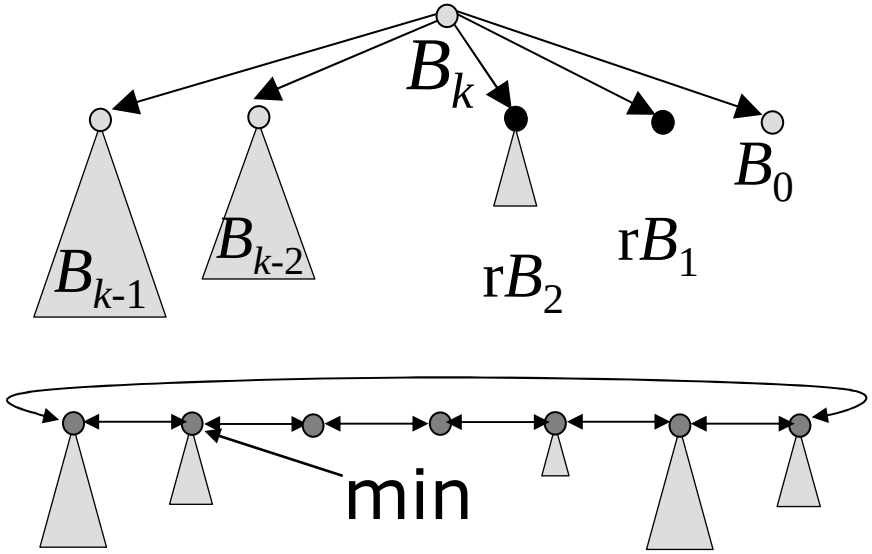
Prove by strong induction: #nodes ≥
$1 + F_1 + F_2 + \ldots + F_{k-1} + F_k$
$= F_{k+2} \geq \varphi^k$

$\varphi = (1+\sqrt{5})/2 > 1.6$

$F_0=0, F_1=1, F_{k+1}= F_k + F_{k-1}$

Lemma: A *relaxed* binomial tree of order k has $\geq F_{k+2} \geq \Omega(1.6^k)$ nodes

Fibonacci Heaps



A ***relaxed*** binomial tree of order $k \geq 1$ is a root with k children: relaxed binomial trees of *distinct* orders.

Operations with amortized costs:

ExtractMin: $O(\log n)$

DecreaseKey: $O(1)$

Merge: $O(1)$

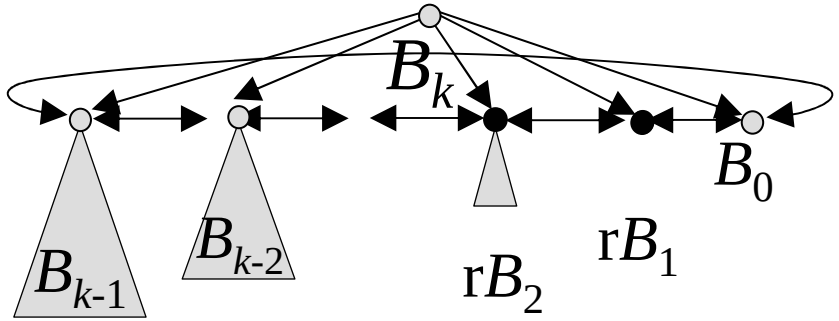
InsertKey: $O(1)$

Create: $O(1)$

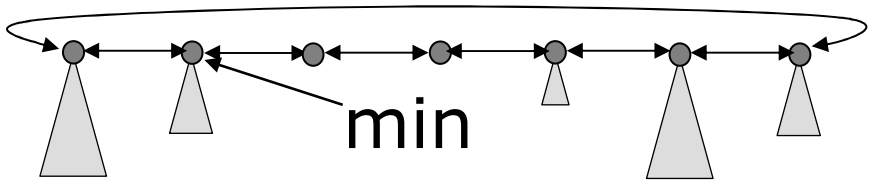
A Fibonacci Heap is a list of t heap-ordered relaxed binom. trees with pointer to the min.

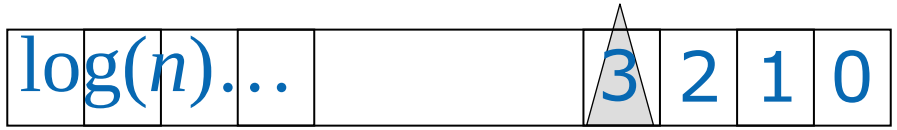
Lemma: A *relaxed* binomial tree of n nodes has order $k \leq O(\log n)$

ExtractMin: O(log *n*) Amortized



A *relaxed* binomial tree of order $k \geq 1$ is a root with k children: relaxed binomial trees of *distinct* orders.





1. Delete target of min.pointer
2. Merge Fibonacci Heaps.
3. **Consolidate** list s.t. relaxed binom.trees have distinct orders k

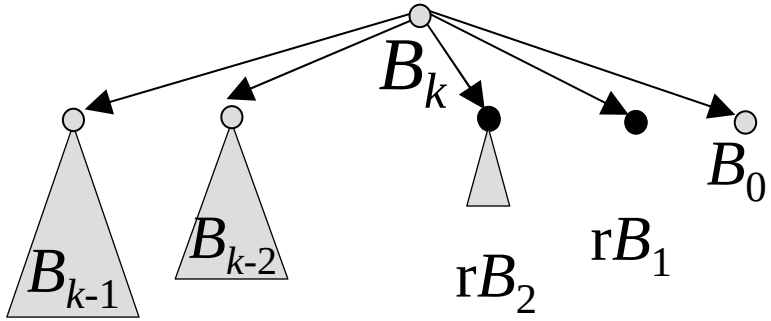
Cmp. adding t binary powers: tot.cost $O(t)$

A Fibonacci Heap is a list of t heap-ordered relaxed binom. trees with pointer to the min.

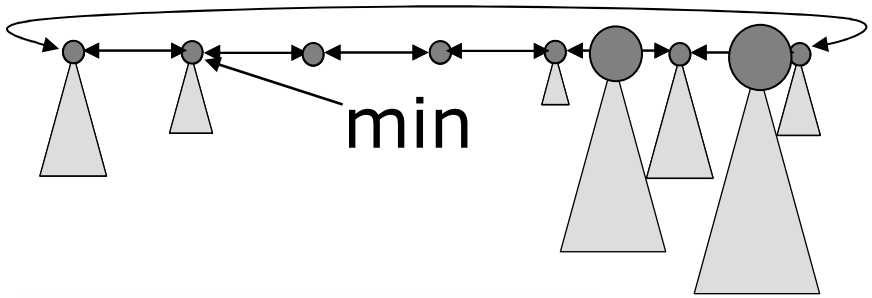
Conceive $\Phi := \Theta(\, t\,) \Rightarrow \Phi_0 = 0 \leq \Phi_n,\ c_j + \Phi_j - \Phi_{j-1} \leq O(\log n)$

$$\sum_{1 \leq j \leq n} c_j / n \ \leq\ \max_j (c_j + \Phi_j - \Phi_{j-1}) + (\Phi_0 - \Phi_n)/n$$

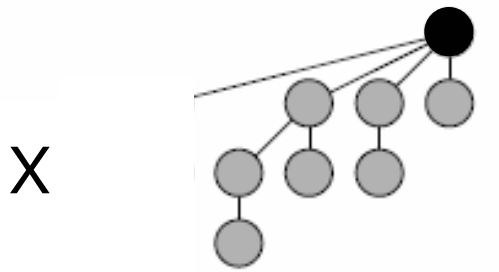
DecreaseKey: O(1) Amortized



A *relaxed* binomial tree of order $k \geq 1$ is a root with k children: relaxed binomial trees of *distinct* orders.



- cut subtree
- **mark** parent
- if already **marked**:
- reset, cut & cascade up



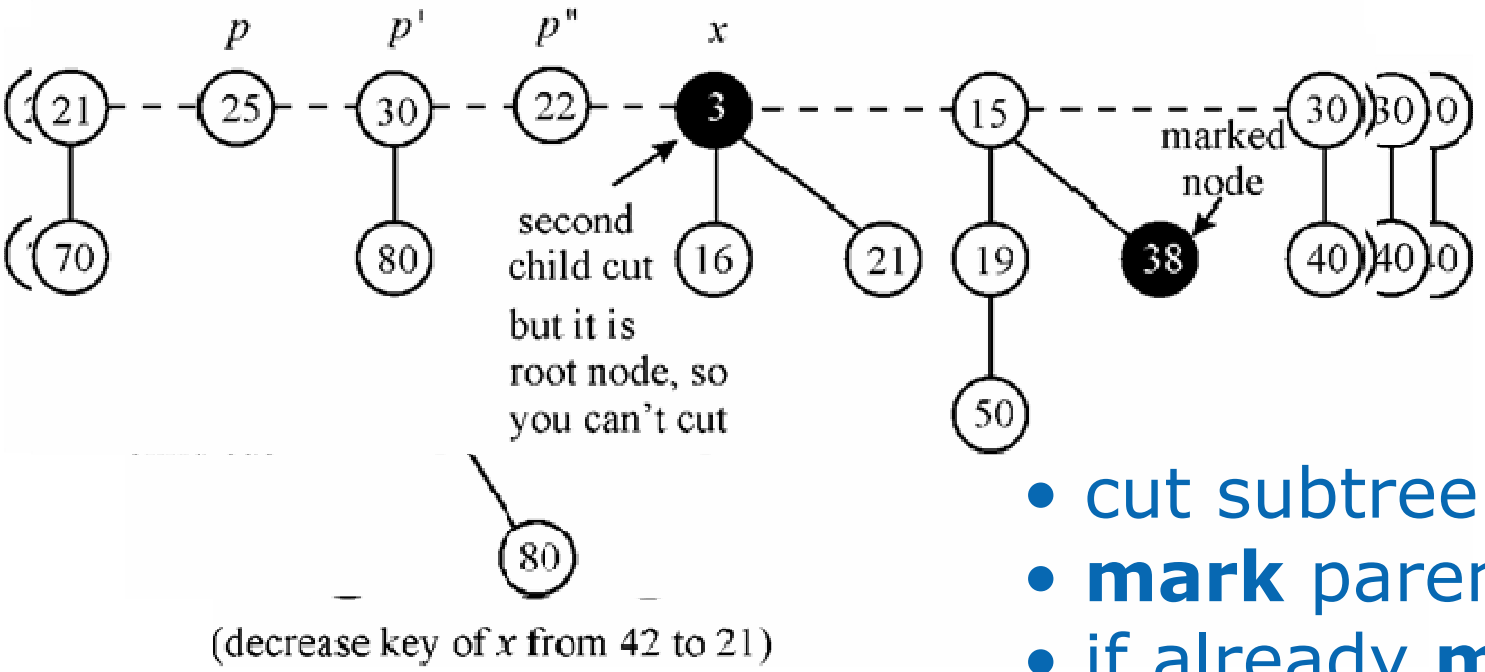
m := #marks

A Fibonacci Heap is a list of ***t*** heap-ordered relaxed binom. trees with pointer to the min.

Conceive $\Phi := \Theta(t+2m)$ $\Phi_0 = 0 \leq \Phi_n$, $c_j + \Phi_j - \Phi_{j-1} \leq O(1)$

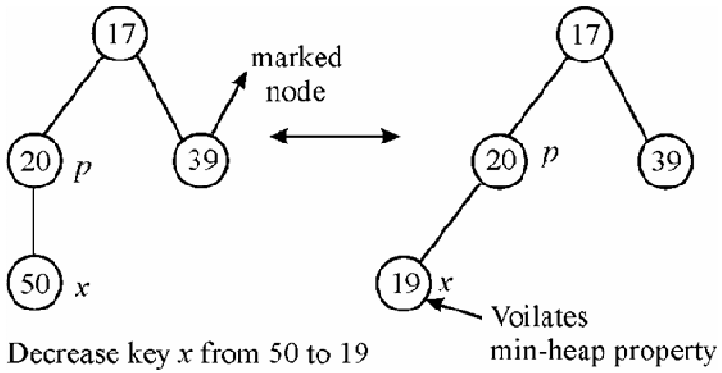
$$\sum_{1 \leq j \leq n} c_j/n \leq \max_j (c_j + \Phi_j - \Phi_{j-1}) + (\Phi_0 - \Phi_n)/n$$

Cuts, Marks, and Cascading

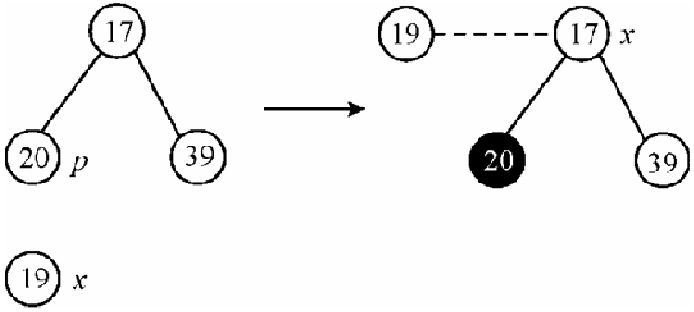


(decrease key of x from 42 to 21)

- cut subtree
- **mark** parent
- if already **marked**:
- reset, cut & cascade up



Decrease key x from 50 to 19



Minimum Spanning Tree

Tree: connected (undirect.) graph without cycles

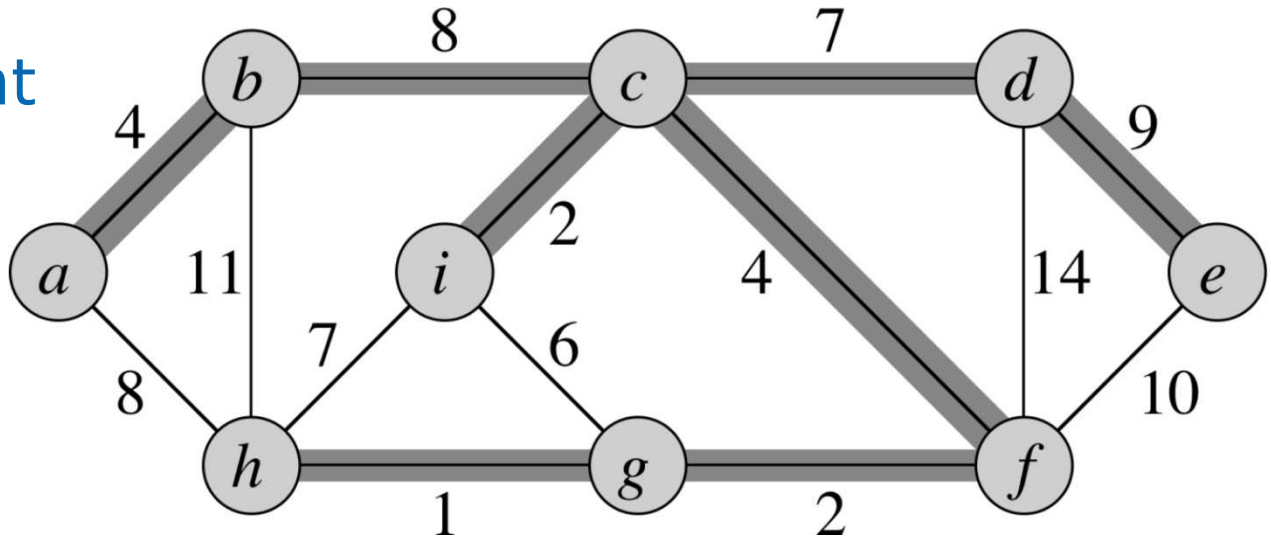
weighted undir. graph (V,E,w) where $E \subseteq V \times V$ and $w: E \to \mathbb{R}$ s.t. $(u,v) \in E \Rightarrow (v,u) \in E \ \wedge \ w(u,v) = w(v,u) < \infty$

MST (Minimum Spanning Tree) of (V,E,w):

a (i) *connected* subset F of the edges E which

(ii) includes *all* vertices and

(iii) has *least* weight $w(F) = \sum_{(u,v) \in F} w(u,v)$ among all $F' \subseteq E$ satisfying (i)+(ii).



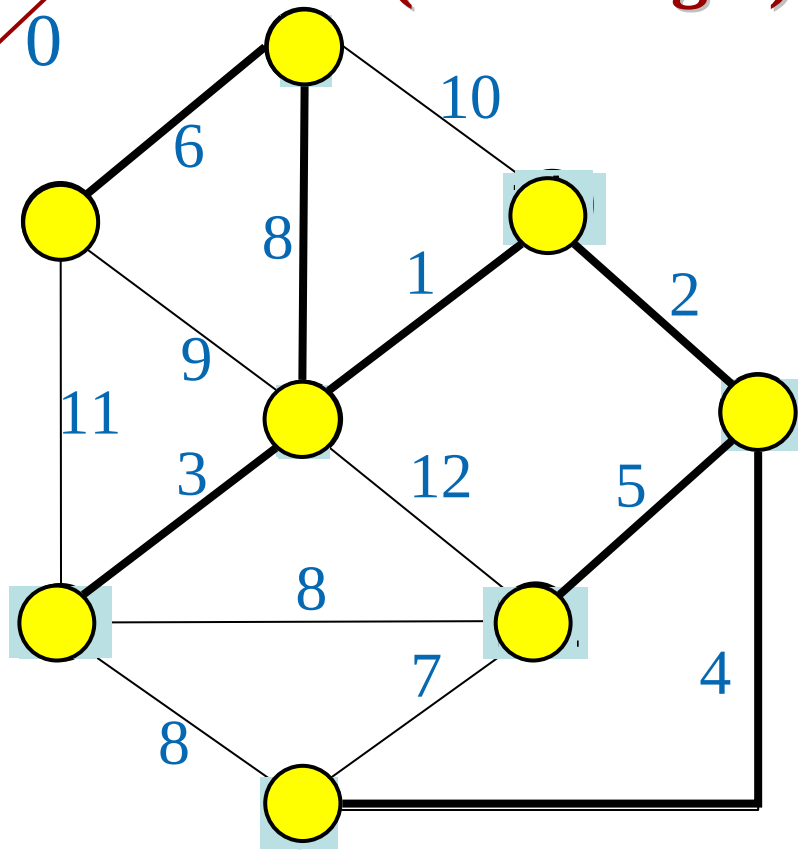
Prim's Algorithm for MST


```
// w : E ⊆ V×V → ℕ   edge weights
T := {};  FOREACH v∈V DO
   v.dist := ∞ ;  v.neighbor := NIL;  DONE
Q := V;  WHILE not Q.isEmpty DO
  u := Q.ExtractMin;   u.dist := 0
  IF u.neighbor ≠ NIL THEN
     T := T ∪ { (u.neighbor , u) } ;
  FOREACH v adjacent to u DO
     IF w(u,v)<v.dist THEN
     v.neighbor := u;
     Q.decreaseKey(v,w(u,v));
     END;
```

$n=|V|$ times

$m=|E| \geq n$ times

Fibonacci Heap:
$O(m + n \cdot \log n)$



Kruskal's Algorithm for MST

// $w : E \subseteq V \times V \to \mathbb{N}$ edge weights

$T := \{\}$; FOREACH $v \in V$ DO **MakeSet**(v); // singletons

$n = |V|$ times

FOREACH edge $(u,v) \in E$ in order of increasing weight DO

IF u and v do NOT belong to the same subset

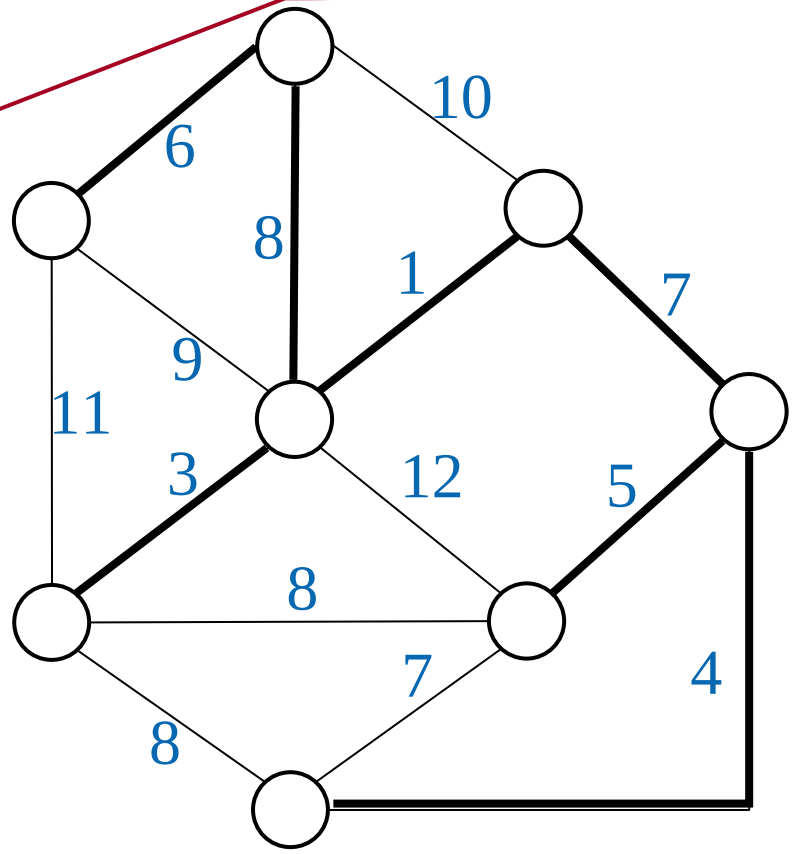
THEN $T := T \cup \{ (u,v) \}$;

Union(u,v);

END;

FindSet$(u) \neq$ **FindSet**(v),

$m = |E|$ times



Disjoint-Set Data Structure:

$O(m \cdot \alpha(m)) < o(m \cdot \log^*(m))$

Prim's Algorithm with
Fibonacci Heap: $O(m + n \cdot \log n)$

Fast&Slowly Growing Functions

2^n exponential $\quad \lceil \log N \rceil = \min\{ n : 2^n \geq N \}$

2^{2^n} doubly exponential $\quad \lceil \log\log N \rceil = \min\{ n : 2^{2^n} \geq N \}$

…

$2^{2^{\cdot^{\cdot^2}}}$ tower of height n

aka tetration $2\uparrow\uparrow n = 2^{2\uparrow\uparrow(n-1)}$

$\log^*(N)$ = #iterations of log before argument ≤ 1
$= 1 + \log^*(\log N), \quad N>1$

Example: $\log(2^{64})=?$ $\quad \log\log(2^{128})=?$ $\quad \log^*(2^{512})=?$

Wilhelm Ackermann (1896—1962) function

$A_0(n)=n+2, \quad A_{k+1}(0)=A_k(1), \quad A_{k+1}(n+1) = A_k(A_{k+1}(n))$

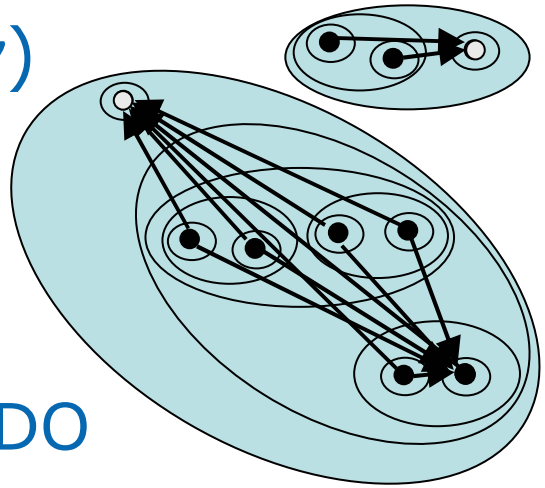
$A_1(n) = 2n+3, \quad A_2(n) = 2^{n+3}-3, \quad A_3(n) = 2\uparrow\uparrow(n+3)-3$

Inverse Ackermann $\alpha(N) = \min\{ n : A_n(n) \geq N \}$

Disjoint-Set Data Structure

MakeSet(x) **FindSet**(x) **Union**(x,y)

return a unique "handle" to the set



Recall Kruskal MST of $G=(V,E,w)$

FOREACH $(u,v)\in E$ in increasing weight w DO

IF **FindSet**$(u)\neq$**FindSet**(v)

THEN $T := T \cup \{ (u,v) \}$; **Union**(u,v) ;

amortized efficiency?

$O(\log n)$

Naïve implementation as forest of depth 1:

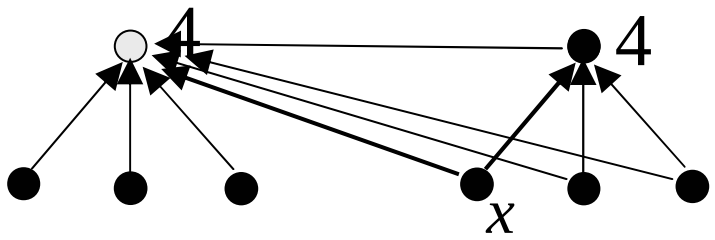
N times alternate **MakeSet+Union** $\rightarrow$ total time $\Omega(N^2)$

Union-by-weight heuristic: attach smaller to larger tree

Theorem: This yields total time $O(m+n\cdot\log n)$ for any sequence of n calls **MakeSet** & m calls **FindSet/Union**

Analysis of Union-by-Weight

`MakeSet`(x) **`FindSet`**(x) **`Union`**(x,y)



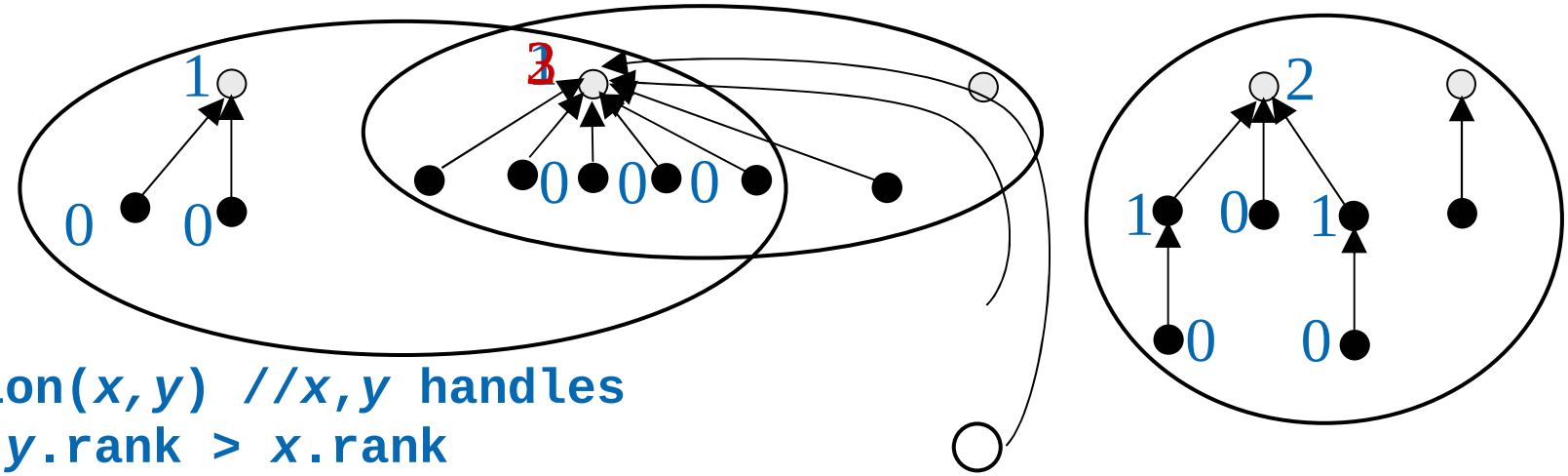
Observation: A node's link gets *updated* only when its set is combined with one of larger or equal size.

So *update #*k can occur only after $n \geq 2^k$ calls **`MakeSet`**.

`MakeSet`, **`FindSet`**: $O(1)$, **`Union`** ?

Theorem: This yields total time $O(m+n \cdot \log n)$ for any sequence of n calls **`MakeSet`** & m calls **`FindSet/Union`**

Lazy Union by Rank, Path Compression




```
Union(x,y) //x,y handles
if y.rank > x.rank
 then attach x to y
else attach y to x;
 if x.rank == y.rank
  then x.rank++; fi; fi;
```

```
function FindSet(x): if x.parent≠NIL
 then x.parent := FindSet(x.parent);
return(x.parent);
```

~~Naïve~~ implementation as forest of unbounded depth:

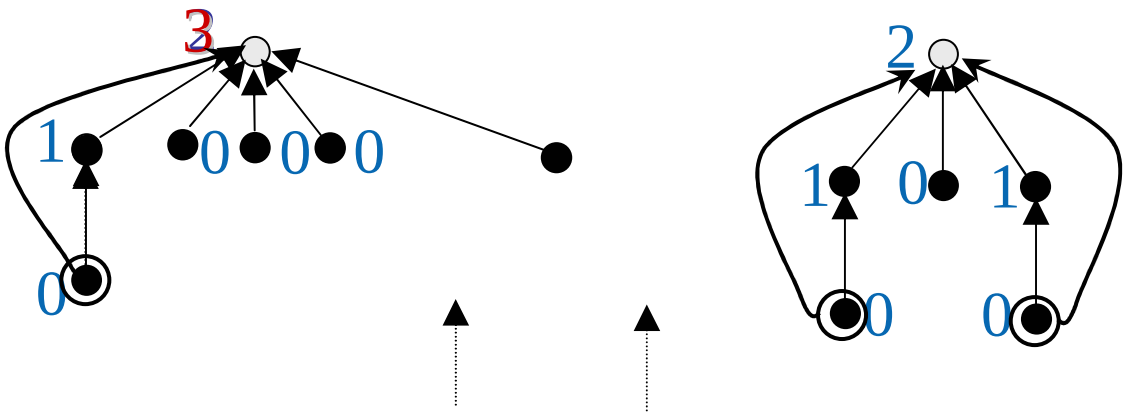
MakeSet, **Union**: $O(1)$, **FindSet** ? *Path compression*

Lazy Union-by-rank: attach lower rank tree to higher

Theorem: This algorithm makes $n\times$ **MakeSet** and $m\times$ **FindSet** and **Union** run in total time $O(m+n\cdot\alpha(n))$.

Separating Union from Find

"semi-
dynamic"



doi:10.1137/S0097539703439088

Re-order: First **MakeSet**s, then **Union**s, then **Find**s

- Rank = height
- *partial* path compression P
- "*closed*": new parent = top(P)
- "*open*": new parent = some ancestor of top(P)

$$\text{cost}(P) := \begin{cases} := \#\text{nodes changing parents} \\ \quad = \text{length} - 1 & \text{for } \textit{closed } P \\ := 0 & \text{for } \textit{open } P \end{cases}$$

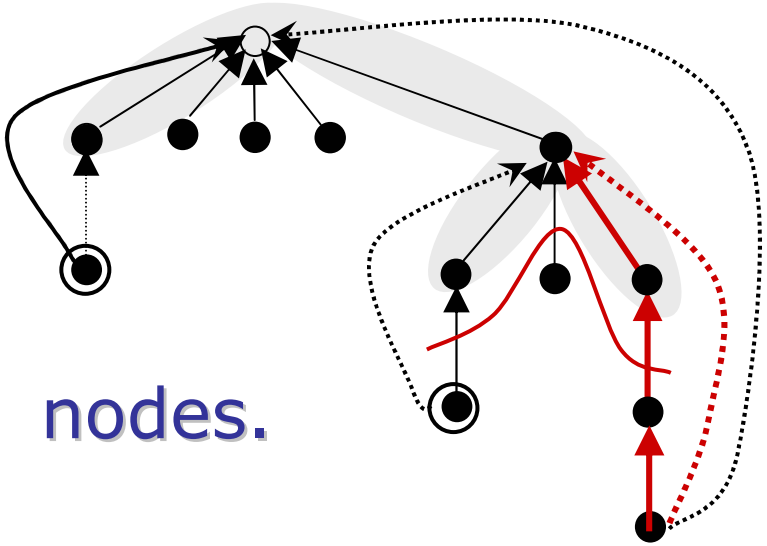
Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial path compressions

$$|\mathcal{P}| := \#\textit{closed } P \in \mathcal{P}$$

$\text{cost}(\mathcal{P})$ = add costs of all $P \in \mathcal{P}$

Cost of m Partial Path Compressions

"*semi-*dynamic"



Fix a forest with $n = |V|$ nodes.

Fix set $T \subseteq V$ containing all its ancestors.

Split every path P into:

- part $P_{T'}$ *out*side of T
- part P_T *in*side of T.

(Each part possibly $\varnothing$)

Lemma:

a) $\text{cost}(\mathcal{P}) \le \text{cost}(\mathcal{P}_T) + \text{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$

b) $|\mathcal{P}_T| + |\mathcal{P}_{T'}| \le |\mathcal{P}|$

- *partial* path compression P
- "*closed*": new parent = top(P)
- "*open*": new parent = some ancestor of top(P)

$\text{cost}(P)$:=
:= #nodes changing parents
= length -1 for *closed* P
:= 0 for *open* P

Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial path compressions

$|\mathcal{P}|$:= #*closed* $P \in \mathcal{P}$

$\text{cost}(\mathcal{P})$ = add costs of all $P \in \mathcal{P}$

First Cost Estimate

$|T|:=n/2 \;\Rightarrow\; \text{cost}(\mathcal{P}) \le (|\mathcal{P}|+|V|/2)\cdot\log_2|V|$
$\le (m+n)\cdot\log n$

Proof by induction + lemma: $\text{cost}(\mathcal{P})$
$\le \text{cost}(\mathcal{P}_T) + \text{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$
$\le (|\mathcal{P}_T|+|T|/2)\cdot\log_2|T|$
$+ (|\mathcal{P}_{T'}|+|T'|/2)\cdot\log_2|T'|$
$+ |\mathcal{P}| + |T'|$
$\le (|\mathcal{P}|+|V|/2)\cdot\log_2|V|$ ■

Fix set $T\subseteq V$ containing all its ancestors.

Split every path P into:

- part $P_{T'}$ *out*side of T
- part P_T *in*side of T.

(Each part possibly $\varnothing$)

Did not take into account *union-by-rank*!

Lemma:

a) $\text{cost}(\mathcal{P}) \le \text{cost}(\mathcal{P}_T) + \text{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$

b) $|\mathcal{P}_T| + |\mathcal{P}_{T'}| \le |\mathcal{P}|$

Forest $\mathcal{F}$ with $n = |V|$ nodes

Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial path compressions

Rank Forests Divide/Conquer

Def: In a *rank forest* of rank $r\in\mathbb{N}$, each node x has $\mathrm{rank}(x)\le r$ and children of all&only ranks $<\mathrm{rank}(x)$.

Lemma: Let $\mathcal{F}$ a rank forest of rank r.

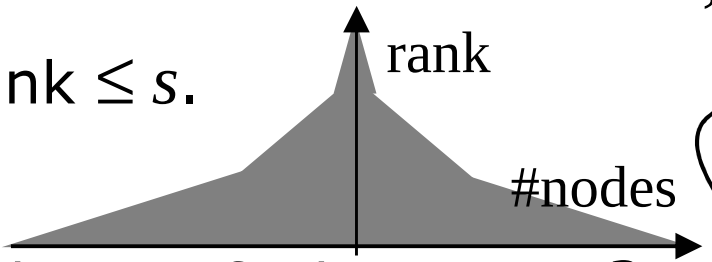
For $s<r$ consider $T := \{\text{nodes of rank}>s\}$

a) $\mathcal{F}\cap T$ is a rank forest of rank $\le r-s-1$,

b) $\mathcal{F}\cap T'$ is of rank $\le s$.

c) $|T| \le |V| / 2^{s+1}$;

$\mathrm{rank}(x)=s \Rightarrow$ subtree of x has size $\ge 2^s$.

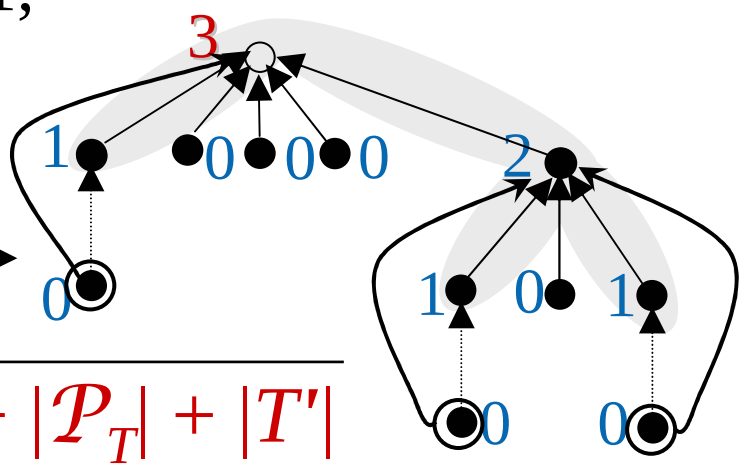


Fix set $T\subseteq V$ containing all its ancestors.

Split every path P into:

- part $P_{T'}$ *out*side of T
- part P_T *in*side of T.

(Each part possibly $\varnothing$)



a) $\mathrm{cost}(\mathcal{P}) \le \mathrm{cost}(\mathcal{P}_T) + \mathrm{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$

b) $|\mathcal{P}_T| + |\mathcal{P}_{T'}| \le |\mathcal{P}|$

Forest $\mathcal{F}$ with $n = |V|$ nodes

Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial path compressions

Union-by-Rank Analyses

Let $s:=\log r$, $t:=\log s$ $\Rightarrow$

$$\text{cost}(\mathcal{P}_{V'_s}) \le n + \text{cost}(\mathcal{P}_{V'_t}) + (|\mathcal{P}_{V'_s}|-|\mathcal{P}_{V'_t}|) + n$$

$\varphi(n,m,r)$:= worst-case cost of m partial paths compressions on any rank forest of rank r with n nodes.

Previous: $\varphi(n,m,r) \le (m+n/2)\cdot\log n$

$\varphi(n,m,r) = \text{cost}(\mathcal{P})$

$\le n + \text{cost}(\mathcal{P}_{V'_s}) + (|\mathcal{P}_V|-|\mathcal{P}_{V'_s}|) + n \le O(m+n\cdot\text{loglog } n)$

$\le 2n + \text{cost}(\mathcal{P}_{V'_t}) + (|\mathcal{P}_V|-|\mathcal{P}_{V'_t}|) + 2n \le O(m+n\cdot\text{logloglog } n)$

$\le k\cdot n + \text{cost}(\mathcal{P}_{V'_k}) + (|\mathcal{P}_V|-|\mathcal{P}_{V'_k}|) + k\cdot n \le O(m+k\cdot n\cdot\log^{(k)} n)$

$k:=\log^* n$

$$\varphi(n,m,r) \le O(m+n\cdot\alpha(n))$$

a) $\text{cost}(\mathcal{P}) \le \text{cost}(\mathcal{P}_T) + \text{cost}(\mathcal{P}_{T'}) + |\mathcal{P}_T| + |T'|$

b) $|\mathcal{P}_T| + |\mathcal{P}_{T'}| \le |\mathcal{P}|$

Forest $\mathcal{F}$ with $n = |V|$ nodes

Sequence $\mathcal{P}=(P_1,..,P_m)$ of partial path compressions

§4 Summary

- Algorithmic Cost Analysis
  - Motivation Bit-Cost of Repeated Increment/Binary Add.
  - Amortized vs. Average vs. Worst-Case Analysis
- Fibonacci Heaps:
  - Relaxed Binomial Trees
  - ExtractMin and DecreaseKey
- Minimum Spanning Tree
  - Prim's Algorithm with Binomial vs. Fibonacci Heap
  - Kruskal's Algorithm: *Union-Find* data type
- Disjoint Set Data Structures
  - Fast and Slowly Growing Functions
  - Analysis of *Union-by-Weight*
  - Lazy Union-by-Rank with Path Compression